A Validation Method Adapted to Dynamic Service Composition

Jacqueline Floch[1,2] and Rolv Bræk[2]

[1]SINTEF Telecom and Informatics N-7465 Trondheim, Norway
E-mail: Jacqueline.Floch@sintef.no

[2]NTNU, Department of Telematics N-7491 Trondheim, Norway
E-mail: {Jacqueline.Floch, Rolv.Bræk}@item.ntnu.no

Abstract

The dynamic composition of services as visioned in a Plug-and Play approach requires new design and validation methods. We propose a service design based on the concept of roles. The feature of composite state newly introduced in SDL 2000 is used to model roles and their composition. We introduce a validation approach tightly integrated with the composition of roles. The approach takes advantage of the system structure. In that way, it will be possible to reuse the results of the analysis done before the modification of the system when a new component is introduced. Furthermore, as components may be bound dynamically at run-time, the analysis is defined such that it can be applied on state machine types - not only instances.

1 Introduction

A traditional approach to service life-cycle as proposed by IN [1] or TINA [2] with the associated scenarios for service deployment is not flexible enough to future market expectations and service providers needs. Users expect to access a similar set of services independently of what network they happen to use, they expect to get access to new and useful services as they become available, and they expect to be able to communicate seamlessly with other users and applications regardless of what service provider and network operator they use. The mobility of users creates new needs for adaptation. Services need to make the most of their surroundings and adapt themselves to overcome any limitations temporarily posed by the current user and network contexts. Building services operating satisfactorily under such conditions poses new challenges and requires new solutions and new engineering methods.

The goal of the PaP project at NTNU is to define a framework for service development and execution that enables services to be designed separately and then composed dynamically using Plug-and-Play techniques [3]. In the frame of the PaP project, our work has focused on two issues: the design of compositional systems and the validation of interactions between components.

Service design is complex. Communication services normally require the coordinated effort of several distributed components, where some of the components may be involved in several services. In a PaP context, this complexity even increases as services should be designed such that they can be dynamically adapted to changing contexts. Our approach to service design is based on service roles [4]. By using services roles, we are able to better comprehend the collaborations between components involved in a service. We are able to break down the complexity of service specification, and to combine roles to provide new services in a flexible way. We describe service roles as state machines, and propose to use SDL 2000 [5] and the newly introduced composite states to model roles.

A major drawback with the current distributed processing approaches and component based development approaches is that they lack support for describing interactions between computational objects. Interfaces are described by static interfaces limited to the declaration of operation signatures. Such interface descriptions do not provide sufficient support for building systems that behave correctly. In our approach we describe interfaces, or association roles, as state machines. Association roles are obtained by projection from service roles. We propose an approach to the validation of association roles. Validation is tightly integrated with the composition of service roles, and can be applied incrementally. The approach enables us to validate parts of systems, and then to apply validation on systems being adapted.

In this paper, we first introduce to the concepts of service roles (s-roles) and service association roles (a-roles). Section 3 presents the modelling of s-roles and their composition. In Section 4, we define the a-roles as projections of s-roles. We also propose a set of transformations that are applied on a-roles in order to simplify the validation analysis, and we identify particular behaviour patterns that require special care during validation. Finally, our validation approach is presented in Section 5.

2 Fundamental concepts

2.1 Service roles

Actors and service roles (s-roles) are key concepts in the PaP framework. A service is seen as a collaboration between s-roles, and service execution requires the assignment of roles to computational objects called actors. The concept of role was already introduced in the end of the 70's in the context of data modelling [6] and has emerged again in the object-oriented literature [7], [8]. In our approach, s-roles encapsulate the functional properties of components involved in a service. They enable us to better comprehend the contribution of a computational object or actor in a service.

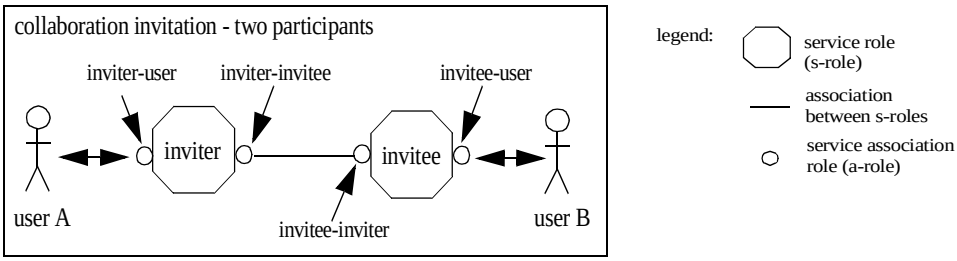

Figure 1. S-roles and a-roles.

S-roles can be decomposed into smaller behavioural elements. For example, the s-role "A-subscriber" in a telephony service may be decomposed into distinct functional elements: inviter, setup and release involved during the different service phases. These elements, as part of collaborations, are themselves elementary s-roles. Conversely, complex s-roles can been produced by composing elementary s-roles.

2.2 Service association roles

S-roles interact with other s-roles over associations. A service association role (a-role) is the visible behaviour of an s-role on an association. A-roles abstract the internal behaviour of s-roles, and the interactions towards other s-roles. This abstraction facilitates the validation analysis.

A-roles overcome the limitations of static object interfaces as defined in CORBA or DCOM. Architectures based on traditional object interfaces lack two main properties [9]. They only describe the functions provided by an object, and fail to describe the functions required by an object. This makes it difficult to determine the effects of changing an interface on other objects. Moreover, they do not describe the semantics of a connection between objects and the constraints on using the interfaces. It is not possible to ensure that the interactions between objects will occur in a correct order. Unlike traditional object interfaces, a-roles describe dialogues or protocols between s-roles. Two a-roles on an association complement each other, and a-roles required by an s-role can be determined from the a-roles that this s-role provides.

3 Service role modelling and composition

Describing the behaviours of individual objects in terms of states and transitions has proven to be of great value, and is widely adopted in most engineering approaches[10]. We use the SDL language to specify s-roles and actors playing s-roles. As SDL does not define the concepts of role and actor, SDL features that fit these concepts have to be selected. As SDL composite states represent parts of behaviour, they are well suited to represent elementary s-roles that also are parts of behaviour. SDL composite states allow structuring state machines, and thus also fit the modelling of composition. Actors are represented using SDL process agents.

Through the composition of s-roles we produce the complete behaviour of an actor in a service. Different forms of composition are applied on s-roles depending on the types of dependencies existing between s-roles. While sequential composition enforces behaviour ordering, concurrent[1] composition supports simultaneous behaviours. Sequential composition encompasses true sequential composition, guarded sequential composition, choice and disabling. Concurrent composition encompasses parallel composition and synchronized composition.

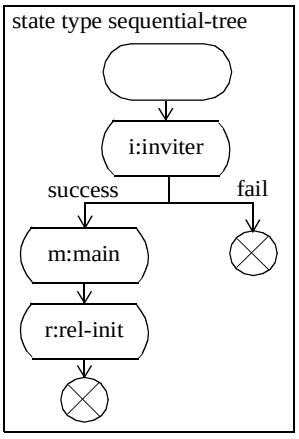

Figure 2. Sequential role composition.

Sequential composition is described in SDL by linking the elementary s-role states in a composite state. The ordering of execution of the composed s-roles is enforced by the definition of the composite state. No major adaptation of the s-roles to be composed is required. Labelled entry and exit points, and entry and exit procedures facilitate the specification of choices and disabling.

1.With concurrency, we do not mean true parallelism, but rather interleaving.

Concurrent composition uses process agents. S-roles are encapsulated within process agents. Process agents may contain other process agents that execute in alternating manner. It is possible to specify several levels of concurrency as process agents may contain other process agents. An alternative to process agents is provided by state aggregation. State aggregation has many limitations, and can only be used to model some cases of static parallel composition of instances of distinct s-roles.

4 Service association roles

A-roles capture the interaction behaviour of an s-role on an association. In order to identify the concepts needed for a-role modelling, it is suitable to think about a-roles as projections of s-roles. A-roles should exhibit the same behaviour as the s-roles they are derived from, on the association they are attached to. This means that an s-role and the projected a-role should be able to generate the same sequences of outputs on an association when offered the same sequence of inputs on this association. The behaviour determining the choice of a behaviour sequence is not visible at association interface, and an a-role may appear to make non-deterministic choices. This non-determinism results from the abstraction of the s-role internal decisions and interactions on other associations.

We describe a-roles as state machines using a notation inspired from SDL. As a-roles restrict to the visible behaviour of s-roles on associations, full SDL is not needed. Some extensions to SDL are introduced in order to abstract non-observable behaviours.

4.1 Projection

As the main purpose of a-roles is to validate the interaction behaviours of an s-role with other s-roles, the projection of s-roles to a-roles is defined such that it maintains the behaviour provided by an s-role on an association. We call this behaviour the observable association behaviour.

The projection of an s-role state graph leads to an a-role state graph. State and transitions are maintained in the derived graph. The visible signals, i.e. the signals sent and received on the association the a-role is attached to, are also maintained. The sending of non-visible signals is not represented, and the consumption of non-visible signals is abstracted to SDL spontaneous inputs. S-role internal actions are not represented in the a-roles. Decision questions, as internal actions, are also hidden. On the other hand, decision choices are represented. The projection of a decision may produce a non-deterministic choice. The projection of transitions may lead to empty spontaneous transitions that we call τ-transitions.

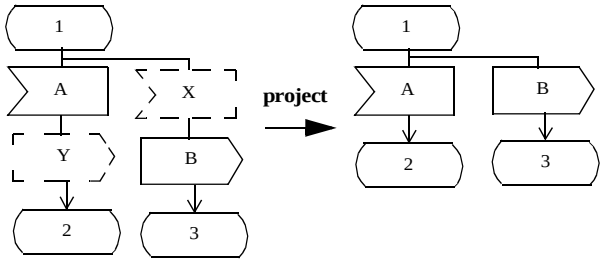

Figure 3. Projection: from s-role to a-role.

The projection of save is complex. As the activation of a spontaneous transition may occur at any time, independently of the presence of signals in the input port, it may anticipate the retrieval of a saved signal from the input port (as shown in Figure 4). In order to define a simple projection of save that maintains the observable association behaviour also when

the saving of visible signals is combined with the consumption of non-visible signals, we propose to constraint the usage of save. Constraints are expressed by design rules.

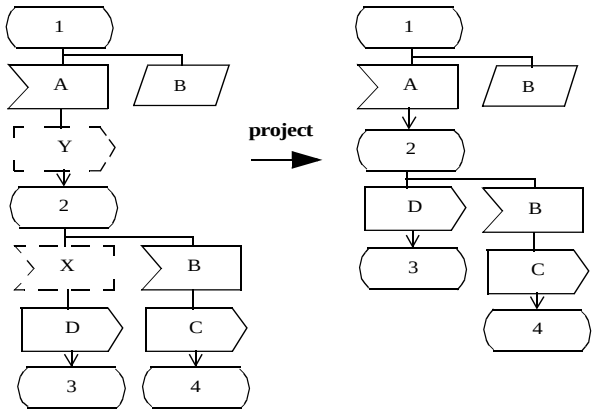

Figure 4. Undesirable save projection behaviour.

We propose two design rules related to using save. One enforces save consistency, i.e. the saving of a signal should be re-iterated in the successor state(s) until the consumption of the saved signal is specified. The other enforces using save either for modelling strict input consumption input orderings or for modelling concurrent behaviours, not both. Using these rules, interferences between spontaneous transitions and the retrieval of a saved signal do not occur. The projection of save is defined such that only the saving of visible signals is represented in the a-role graph.

In our work, we have shown that the proposed s-role projection maintains the observable association behaviour.

4.2 A-role graph refinement

In order to simplify interface validation, we introduce three transformations to be applied on the a-role graph. These transformations facilitate the generation of consistent complementary a-roles and other validation operations, and they reduce the size of the graph. We have shown that the transformations maintain the observable association behaviour.

4.2.1 Transformation to a transition chart

We first transform the a-role state graph to a transition chart. A transition chart is a particular state graph where transitions between states are attached a single event: an input, an output or a silent event called the τ-event. τ-events trigger τ-transitions. We define the σ-state as a specialization of the SDL state: a σ-state implicitly save all visible signals. The transformation of an a-role state graph to a transition chart is performed by inserting a σ-state before the sending of a signal, when no state already precedes signal sending.

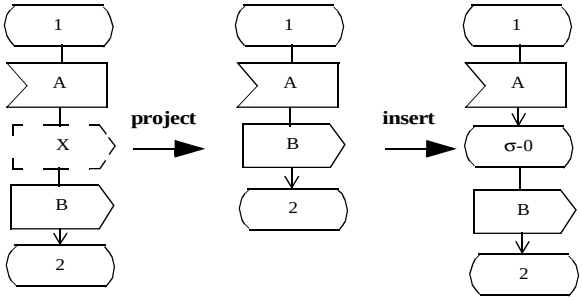

Figure 5. σ-state insertion.

4.2.2 Gathering

The transformation from s-roles to a-roles may lead to graphs where several states linked by τ-transitions take place successively. In some cases, these τ-transitions have no influence on the observable association behaviour. Gathering is a transformation that replaces such states by a single state, and in that way to reduce the size of a transition chart.

Gathering can only be applied when:

- Any signal specified as an input in the τ-successor[1] is either specified as an input or as a save signal in the τ-predecessor. In the former case, the τ-successor and τ-predecessor should transit to similar successor states. In the later case, no other input should be should be specified in the τ-predecessor.
- Any signal specified as a save in the τ-successor is either specified as an input or as a save signal in the τ-predecessor. In the former case, no other input should be should be specified in the τ-successor.

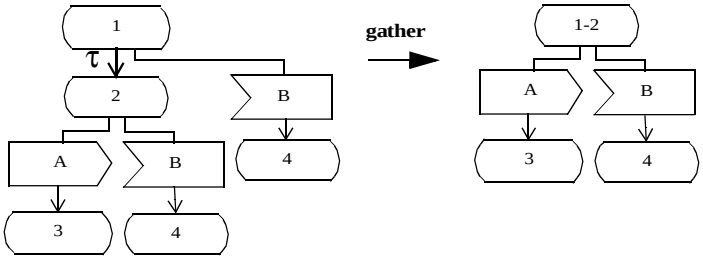

Figure 6. Gathering.

4.2.3 Minimization

Transition charts may contain equivalent states that exhibit the same observable association behaviour and lead to states that also exhibit the same observable association behaviour. In order to facilitate interface validation, it is desirable to replace such states by a single state. This replacement, called minimization, reduces the size of the state graph.

Equivalence may be defined in different manners depending on whether τ-events are observed or not [11]. In our approach, we retain τ-events that contribute to state changes when these changes influence the visible association behaviour. As gathering is a transformation that removes non-observable τ-transitions, we combine gathering and minimization in order to remove redundant and non-observable behaviour.

We have defined equivalence in terms of triggering events such that it is possible to easily define an operational minimization algorithm. An algorithm based on the concept of partitions of k-equivalent states [12] is proposed. The algorithm applies to deterministic and non-deterministic machines.

4.3 Ambiguous and conflicting behaviours

An ambiguous behaviour takes place when an external observer is not able to determine which behaviour is expected by an a-role. A conflict occurs when the behaviours of an a-role and its complementary a-role diverge. Ambiguous and conflicting behaviours will require special care during interface validation. We have identified particular specification patterns that lead to such behaviours.

1.The terms τ-*successor* and τ-*predecessor* are used to denote respectively the successor of a state triggered by a τ-event and the predecessor state of a τ-successor.

4.3.1 Equivoque transitions

Two or more transitions are equivoque when they are defined for the same state and the same event (i.e input, output or τ-event), and lead to distinct non-equivalent states. Equivoque transitions may lead to ambiguous behaviours.

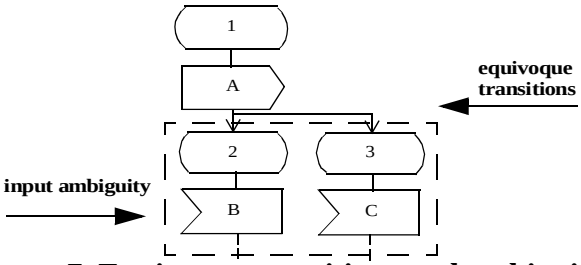

Figure 7. Equivoque transitions and ambiguity.

We distinguish between several types of ambiguity:

- Strong input ambiguity occurs when at some stage of an interaction, an external observer is not able to determine any of the input(s) expected by the a-role state machine.
- Strong mixed ambiguity occurs when at some stage of an interaction, an external observer is not able to determine any of the input or output events expected by the a-role state machine.
- Weak input and mixed ambiguities are special forms of input and mixed ambiguities where some, but not all, events can be observed.
- Termination ambiguity occurs when at some stage of an interaction, an external observer is not able to determine whether the a-role state machine has terminated, or is waiting for a triggering event to occur.
- Termination occurrence and termination condition ambiguities are forms of termination ambiguity where termination can be observed, but not the occurrence or condition of termination.

4.3.2 Mixed initiatives

A mixed initiative state is a state where both signal consumption and sending can occur. As a-roles communicate asynchronously, they perceive the occurrence of communication at different moments of time. The reception of a signal is perceived some time after its sending. When an a-role and its complementary a-role are both enabled to send a signal during the same interaction step, the signals sent may cross each other. Such behaviour may lead to unspecified signal reception, and deadlocks where each a-role state machine waits for the other machine's answer.

A-roles and the s-roles they are derived from should be specified such that potential conflicts are detected and resolved.

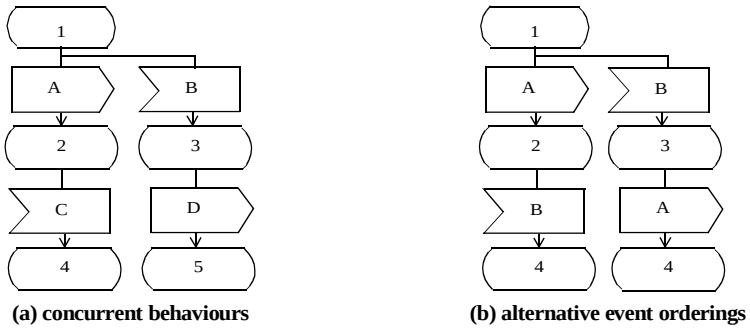

Figure 8. Mixed initiatives.

4.3.3 Acute τ-transitions

Acute τ-transitions are τ-transitions that cannot be removed from the a-role transition chart by gathering and minimization. Acute τ-transitions require special attention: they are a symptom for ambiguity. Acute τ-transitions lead to ambiguous behaviours, either as triggers of equivoque transitions as explained above, or when combined with other transitions.

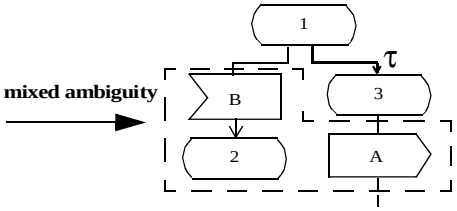

Figure 9. Acute τ-transitions and ambiguity.

5 Validation

The dynamic composition of systems in a PaP context sets requirements on validation:

- The analysis should take advantage of the system structure. If one component is replaced, it should be possible to reuse the results of the analysis done before the modification of the system.
- As components may be bound dynamically at run-time, it should be possible to apply the analysis on state machine types - not instances.

We have focused on safety properties i.e. avoiding that bad behaviours, such as deadlocks, occur. We do not provide any formalism for expressing correctness requirements as done in [13]. Instead we provide validation techniques for avoiding significant interaction errors. We say that roles (a-roles or s-roles) interact consistently when their interactions do not lead to unspecified signal receptions, deadlocks, or improper terminations.

[14] distinguishes between constructive methods that aim to generate the right systems, and corrective methods that aim to detect and correct the errors that are made. Our approach is twofold. It provides support for producing correct specifications, and for detecting errors when checking a system, eventually at run time.

Our major concern is that the validation approach should be easy to understand and perform. The approach should not require detailed knowledge of formal verification techniques from the system developer/tester. As the validation problem is complex, simplification is required. We propose two simplification schemes:

- On one hand, we tightly integrate the validation approach with the composition of s-roles, and propose an incremental validation: elementary s-roles are first validated, then their composition is validated.
- On the other hand, simplification is achieved by making use of projections [15]. The analysis concentrates on dependant behaviours, i.e. on the interactions between s-roles. We seek to make a-roles consistent.

5.1 Specifying dual a-roles

A dual a-role is a complementary a-role of a given a-role, that interacts consistently with this given a-role. A constructive validation approach seeks to produce a dual a-role from a particular a-role.

Mirroring is a transformation that maintains the structure of the graph, and transforms inputs to outputs and outputs to inputs. In our work, we have shown that dual a-roles can be produced by mirroring providing that the initial a-role does not define

- any equivoque transitions,
- any acute τ-transition,
- any mixed initiatives state.

The execution of the a-role and complementary a-role should be coordinated in the case several initial states are defined, i.e. the machines should be entered using consistent entry conditions.

5.1.1 Equivoque transitions

Mirroring fails to produce dual a-roles when applied to a-roles that contain equivoque transitions. When the a-roles do not present any behaviour ambiguity, except termination condition or occurrence ambiguities, the machine can be first transformed by merging before mirroring. Otherwise the s-role which the a-role is derived from should be re-defined.

Merging is a transformation that removes equivoque transitions from a machine. Merging replaces distinct states that are reachable from a state triggered by equivoque transitions through the same sequence of events, by a new single state reachable from the state triggered by the equivoque transition through the same sequence of events; the new state exhibits the behaviour of the merged states. Merging is illustrated in Figure 10. Note that, in that case, the machine contains equivoque transitions but does not present any ambiguity. An external observer perceives which behaviour is selected when receiving “C” or “D”.

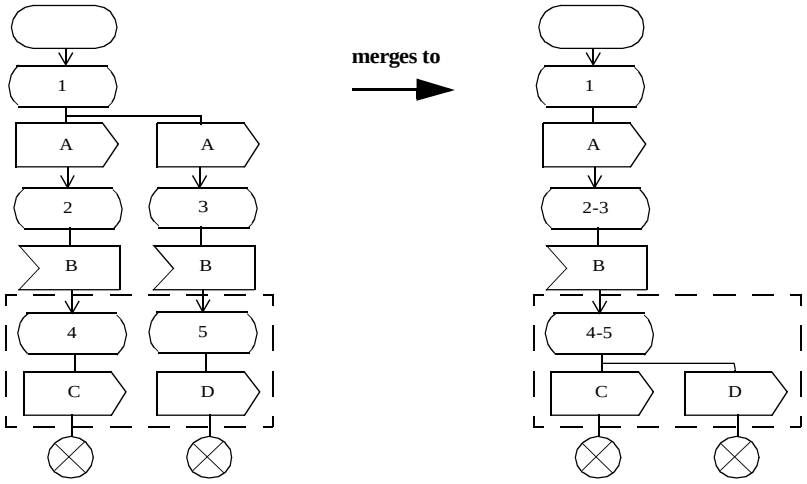

Figure 10. Merging.

In the case the machine presents a strong behaviour ambiguity, an external observer cannot determine the further behaviour, and thus cannot act. The machine should be re-designed. Merging can be applied when re-designing an a-role that presents input or mixed ambiguity. Merging should not be applied on machines that present termination ambiguity, this because exit states have a particular semantics (the machine stops), and should not be merged with non-exit states.

In the case the machine presents a weak behaviour ambiguity, an external observer can only partially determine the further behaviour, it is possible to produce dual a-role that provides a partial behaviour expected by the initial a-role. As our definition of interaction consistency does not address non-executable transitions, the machines interact consistently. However we also advice re-design in that case.

In our work, we have defined algorithms for identifying the various types of ambiguity, and for merging.

5.1.2 Mixed initiative states

If two associated a-roles take the initiative to send a signal simultaneously, the signals may cross each other, and the a-roles perceive the order of occurrence of events differently. The signals they send are received in the states triggered by signal sending in the other machines. In order to avoid unspecified signal reception, the signals specified as inputs in mixed initiative states should be specified as inputs in the states triggered by signal sending in the mixed initiative states. This property is called input consistency. Two a-roles involved in a mixed initiative may interact in a non-consistent manner if their machines are not input consistent.

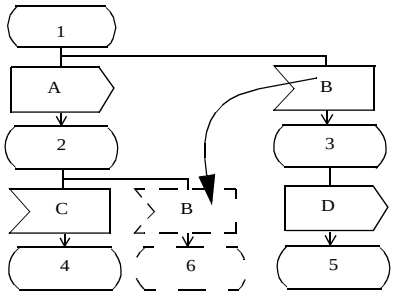

Figure 11. Input consistency.

In addition to enforcing input consistency, mixed initiative require the machines to be designed according to particular rules. Mixed initiatives may either represent concurrent behaviours or alternative event orderings (see Figure 8). These two forms require slightly different rules:

- In the case of concurrent behaviours, conflicts may occur. State machines should be specified such that conflicts can be detected and resolved. We propose design patterns for the resolution of conflicts.
- Alternative event orderings should lead to a common state. In order to facilitate the analysis, we advice to restrict the event sequence to a small number of events.

In order to keep the detection of conflicts simple, we recommend to avoid using mixed initiative states to describe both concurrent behaviours and alternative event orderings.

5.1.3 Acute τ-transitions

When they lead to ambiguous behaviours, acute τ-transitions require the re-design of the s-role machine similarly to equivoque transitions. In addition, we propose design rules that enable remaining τ-transitions to removed from the a-role state graph: the s-role should be made input and save consistent. Finally, describe transformations that enable a dual a-role to be generated even though some τ-transitions are not removed.

5.2 Checking the consistency of two a-roles

A corrective validation method seeks to check the consistency of two a-roles. At consistency checking, we require the a-roles to enforce the design rules introduced for the specification of dual a-roles. No assumption about the complementary a-roles are made. In that way, a-roles are not dependent on a particular behaviour of their complementary a-roles, and these complementary a-roles may easily be changed. The introduction of the save fea-

ture forces us to introduce a consistency checking algorithm that is cognate to the algorithms used for the generation of global state graphs. However our algorithm differs from those algorithms in that it does not use any message queue, but a save queue. Both merging and the design rules applied on s-roles and a-roles contribute to simplify the analysis and to maintain the number of states in the global state low.

5.3 Validating composite roles

Sequential composition of s-roles is specified using the same mechanisms as the specification of elementary s-roles, the techniques proposed for the validation of elementary s-roles also apply to sequentially composed s-roles. In addition, new rules are introduced in order to avoid the non-coordinated start of execution of the s-roles and unspecified signal reception.

Concurrent composition introduces new associations that are validated separately.

6 Related work

Using roles and role collaborations have been proposed in order to enhance code reuse and adaptation in [16], [17]. The approaches focus on programming languages rather than design languages. The descriptions contain many details and the lack of abstraction make them difficult to understand. In [18], an SDL based composition approach is proposed. The authors have however experimented with an earlier version of SDL, and introduced a new notation for modelling composition. SDL 2000 has been recently introduced, and, as far as we know, no work related to the use of SDL 2000 has been published to this day.

Using abstractions in order to reduce the complexity of validation is a usual approach. A theoretical approach is proposed in [19]. Our approach is more pragmatic, and addressed to engineers. In [20], the authors point out the lack of work related to abstracting state machines or SDL. Their approach concentrates on the data part while ours on the control part.

7 Conclusion

This paper has presented an approach to the design compositional systems based on the concept of roles. Roles and their composition are using SDL. However, UML 2.0 that ought to be available soon will probably become an alternative notation [21]. We expect this new version of UML to provide a complete semantics for state machines. Both SDL and UML allow to structure state machines in a similar way as Harel's statecharts [22].

The paper has introduced an approach to the validation of interactions between components. The approach may either apply to elementary or composite roles. In the later case it assumes that components are designed according to the role based design approach. Validation is simplified through the use of projections. We believe that description of component interfaces as association roles is of relevance for distributed processing approaches. A-role descriptions overcome the limitations of static object interfaces. They enable to process the interfaces functions required by a component, and they describe the semantics of an interaction.

Acknowledgments

This work has been done in the frame of the Plug-and-Play project supported by the Norwegian Foundation for Research (NFR).

References

[1] ITU-T. 1997. Intelligent Network. *Introduction to Intelligent Network Capability Set 2*. Recommendation Q.1221.

[2] Berndt, H. and al. 1999. The TINA book: a co-operative solution for a competitive world. Prentice Hall Europe.

[3] F.A. Aagesen and al. 1999. “Towards a Plug and Play Architecture for Telecommunications,” in Proc. of the Fifth International Conference on Intelligence in Networks.

[4] R. Bræk. 1999. “Using Roles with Types and Objects For Service Development,” in Proc. of the Fifth International Conference on Intelligence in Networks.

[5] ITU-T. 1999a. *SDL-2000 Specification and Description Language (SDL)*. ITU-T Recommendation Z.100 (11/99).

[6] C.W. Bachman and M. Daya. 1977. “The role concept in data models,” in Proc. of the Third International Conference on Very Large Databases.

[7] T. Reenskaug and al. 1992. “OORASS: Seamless support for the creation and maintenance of object oriented systems.” Journal of object-oriented programming.

[8] B.B. Kristensen and K. Østerbye. 1996. “Roles: Conceptual6. Abstraction Theory and Practical Language Issues,” Theory and Practice of Object Systems, Vol. 2(3).

[9] Luckham, D.C, Vera, J., and Meldal, S. 1995. Three Concepts of Architecture. *Stanford University Technical Report CSL-TR-95-674*.

[10] Bræk, R. 2000. On Methodology Using the ITU-T Languages and UML. *Telekronikk*, vol. 2, no. 4, pp. 96-106, Telenor. ISSN 0085-7130.

[11] Milner, R. 1989. *Communication and Concurrency.* Prentice Hall.

[12] Hennie, F.C. 1968. *Finite-state models for logical machines*. John Wiley & Sons.

[13] Holzmann, G.J. 1991. *Design and Validation of Computer Protocols*. Prentice Hall.

[14] Bræk, R and Haugen, Ø. 1993. *Engineering Real Time Systems*. Prentice Hall

[15] Lam, S.S., and Shankar, A.U. 1984. Protocol Verification via Projections. *IEEE Transactions on Software Engineering*, vol. 10, no. 4, pp. 325-342.

[16] Mezini, M., and Lieberherr, K. 1998. Adaptative Plug-and-Play Components for Evolutionary Software Development. *Proceedings of OOPSLA’ 98*, ACM SIGPLAN Notices, vol. 33, no. 10, pp. 97-116, ACM Press.

[17] Michael VanHilst and David Notkin. 1996. Using Role Components to Implement Collaboration-Based Designs. *Proceedings of OOPSLA’ 96*, ACM SIGPLAN Notices, vol. 28, no. 10, ACM Press.

[18] RöBler, B., Geppert, B., and Gotzheim, R. 2001. Collaboration-based Design of SDL Systems. In Proc. of the 10th SDL Forum.

[19] Loiseaux, C. and al. 1995. Property Preserving Abstractions for the Verification of Concurrent Systems. In Formal Methods in System Design, Vol. 6, pp. 11-44.

[20] Boroday, S. and al. 2002. Techniques for Abstracting SDL Specifications. To appear in Proc. of SAM’ 2002.

[21] OMG. *Unified Modeling Language Specification*. Version 2.0. To be published.

[22] Harel, D. 1987. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, vol. 8, no. 3, pp. 231-274, Elsevier.